04 Amazon Fine Food Reviews Analysis_NaiveBayes

May 27, 2019

1 Amazon Fine Food Reviews Analysis

Data Source: https://www.kaggle.com/snap/amazon-fine-food-reviews

EDA: https://nycdatascience.com/blog/student-works/amazon-fine-foods-visualization/

The Amazon Fine Food Reviews dataset consists of reviews of fine foods from Amazon.

Number of reviews: 568,454 Number of users: 256,059 Number of products: 74,258 Timespan: Oct 1999 - Oct 2012 Number of Attributes/Columns in data: 10

Attribute Information:

1. Id
2. ProductId - unique identifier for the product
3. UserId - unqiue identifier for the user
4. ProfileName
5. HelpfulnessNumerator - number of users who found the review helpful
6. HelpfulnessDenominator - number of users who indicated whether they found the review helpful or not
7. Score - rating between 1 and 5
8. Time - timestamp for the review
9. Summary - brief summary of the review
10. Text - text of the review

Objective: Given a review, determine whether the review is positive (rating of 4 or 5) or negative (rating of 1 or 2).

[Q] How to determine if a review is positive or negative? [Ans] We could use Score/Rating. A rating of 4 or 5 can be cosnidered as a positive review. A rating of 1 or 2 can be considered as negative one. A review of rating 3 is considered nuetral and such reviews are ignored from our analysis. This is an approximate and proxy way of determining the polarity (positivity/negativity) of a review.

2 [1]. Reading Data

2.1 [1.1] Loading the data

The dataset is available in two forms 1. .csv file 2. SQLite Database

In order to load the data, We have used the SQLITE dataset as it is easier to query the data and visualise the data efficiently.

Here as we only want to get the global sentiment of the recommendations (positive or negative), we will purposefully ignore all Scores equal to 3. If the score is above 3, then the recommendation wil be set to "positive". Otherwise, it will be set to "negative".

In [1]:
```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")


import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle

from tqdm import tqdm
import os
```

In [2]:
```
# using SQLite Table to read data.
con = sqlite3.connect('database.sqlite')

# filtering only positive and negative reviews i.e.
# not taking into consideration those reviews with Score=3
# SELECT * FROM Reviews WHERE Score != 3 LIMIT 500000, will give top 500000 data point
# you can change the number to any other number based on your computing power

# filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 5
```

```
# for tsne assignment you can take 5k data points

# 100000 data points suggested for Naive Bayes.
filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 ORDER BY T


# Give reviews with Score>3 a positive rating(1), and reviews with a score<3 a negativ
def partition(x):
    if x < 3:
        return 0
    return 1

#changing reviews with score less than 3 to be positive and vice-versa
actualScore = filtered_data['Score']
positiveNegative = actualScore.map(partition)
filtered_data['Score'] = positiveNegative
print("Number of data points in our data", filtered_data.shape)
filtered_data.head(3)
```

```
Number of data points in our data (100000, 10)
```

```
Out[2]:      Id   ProductId          UserId    ProfileName  HelpfulnessNumerator  \
        0    10  B00171APVA  A21BT40VZCCYT4  Carol A. Reed                     0
        1  1089  B004FD13RW    A1BPLP0BKERV           Paul                     0
        2  5703  B009WSNWC4   AMP7K1O84DH1T           ESTY                     0

           HelpfulnessDenominator  Score        Time           Summary  \
        0                       0      1  1351209600  Healthy Dog Food
        1                       0      1  1351209600    It is awesome.
        2                       0      1  1351209600         DELICIOUS

                                                        Text
        0  This is a very healthy dog food. Good for thei...
        1  My partner is very happy with the tea, and is ...
        2  Purchased this product at a local store in NY ...
```

```
In [4]: display = pd.read_sql_query("""
        SELECT UserId, ProductId, ProfileName, Time, Score, Text, COUNT(*)
        FROM Reviews
        GROUP BY UserId
        HAVING COUNT(*)>1
        """, con)
```

```
In [5]: print(display.shape)
        display.head()
```

```
(80668, 7)
```

Out[5]:
```
                UserId   ProductId                ProfileName        Time  Score  \
0  #oc-R115TNMSPFT9I7  B007Y59HVM                    Breyton  1331510400      2
1  #oc-R11D9D7SHXIJB9  B005HG9ET0  Louis E. Emory "hoppy"  1342396800      5
2  #oc-R11DNU2NBKQ23Z  B007Y59HVM          Kim Cieszykowski  1348531200      1
3  #oc-R11O5J5ZVQE25C  B005HG9ET0              Penguin Chick  1346889600      5
4  #oc-R12KPBODL2B5ZD  B007OSBE1U    Christopher P. Presta  1348617600      1

                                                Text  COUNT(*)
0  Overall its just OK when considering the price...         2
1  My wife has recurring extreme muscle spasms, u...         3
2  This coffee is horrible and unfortunately not ...         2
3  This will be the bottle that you grab from the...         3
4  I didnt like this coffee. Instead of telling y...         2
```

In [6]:
```
display[display['UserId']=='AZY10LLTJ71NX']
```

Out[6]:
```
              UserId   ProductId                       ProfileName        Time  \
80638  AZY10LLTJ71NX  B006P7E5ZI  undertheshrine "undertheshrine"  1334707200

       Score                                               Text  COUNT(*)
80638      5  I was recommended to try green tea extract to ...         5
```

In [7]:
```
display['COUNT(*)'].sum()
```

Out[7]:
```
393063
```

3 [2] Exploratory Data Analysis

3.1 [2.1] Data Cleaning: Deduplication

It is observed (as shown in the table below) that the reviews data had many duplicate entries. Hence it was necessary to remove duplicates in order to get unbiased results for the analysis of the data. Following is an example:

In [8]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND UserId="AR5J8UI46CURR"
ORDER BY ProductID
""", con)
display.head()
```

Out[8]:
```
       Id   ProductId         UserId      ProfileName  HelpfulnessNumerator  \
0   78445  B000HDL1RQ  AR5J8UI46CURR  Geetha Krishnan                     2
1  138317  B000HDOPYC  AR5J8UI46CURR  Geetha Krishnan                     2
2  138277  B000HDOPYM  AR5J8UI46CURR  Geetha Krishnan                     2
3   73791  B000HDOPZG  AR5J8UI46CURR  Geetha Krishnan                     2
4  155049  B000PAQ75C  AR5J8UI46CURR  Geetha Krishnan                     2
```

```
   HelpfulnessDenominator  Score        Time  \
0                       2      5  1199577600
1                       2      5  1199577600
2                       2      5  1199577600
3                       2      5  1199577600
4                       2      5  1199577600

                            Summary  \
0  LOACKER QUADRATINI VANILLA WAFERS
1  LOACKER QUADRATINI VANILLA WAFERS
2  LOACKER QUADRATINI VANILLA WAFERS
3  LOACKER QUADRATINI VANILLA WAFERS
4  LOACKER QUADRATINI VANILLA WAFERS

                                                Text
0  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
1  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
2  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
3  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
4  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
```

As it can be seen above that same user has multiple reviews with same values for HelpfulnessNumerator, HelpfulnessDenominator, Score, Time, Summary and Text and on doing analysis it was found that ProductId=B000HDOPZG was Loacker Quadratini Vanilla Wafer Cookies, 8.82-Ounce Packages (Pack of 8) ProductId=B000HDL1RQ was Loacker Quadratini Lemon Wafer Cookies, 8.82-Ounce Packages (Pack of 8) and so on

It was inferred after analysis that reviews with same parameters other than ProductId belonged to the same product just having different flavour or quantity. Hence in order to reduce redundancy it was decided to eliminate the rows having same parameters.

The method used for the same was that we first sort the data according to ProductId and then just keep the first similar product review and delelte the others. for eg. in the above just the review for ProductId=B000HDL1RQ remains. This method ensures that there is only one representative for each product and deduplication without sorting would lead to possibility of different representatives still existing for the same product.

```
In [9]: #Sorting data according to ProductId in ascending order
        sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=Fals
```

```
In [10]: #Deduplication of entries
         final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep=
         final.shape
```

```
Out[10]: (71551, 10)
```

```
In [11]: #Checking to see how much % of data still remains
         (final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

```
Out[11]: 71.551
```

Observation:- It was also seen that in two rows given below the value of HelpfulnessNumerator is greater than HelpfulnessDenominator which is not practically possible hence these two rows too are removed from calcualtions

```
In [12]: display= pd.read_sql_query("""
         SELECT *
         FROM Reviews
         WHERE Score != 3 AND Id=44737 OR Id=64422
         ORDER BY ProductID
         """, con)

         display.head()
```

```
Out[12]:       Id   ProductId          UserId               ProfileName  \
         0  64422  B000MIDROQ  A161DK06JJMCYF  J. E. Stephens "Jeanne"
         1  44737  B001EQ55RW  A2V0I904FH7ABY                       Ram

            HelpfulnessNumerator  HelpfulnessDenominator  Score        Time  \
         0                     3                       1      5  1224892800
         1                     3                       2      4  1212883200

                                               Summary  \
         0            Bought This for My Son at College
         1  Pure cocoa taste with crunchy almonds inside

                                                          Text
         0  My son loves spaghetti so I didn't hesitate or...
         1  It was almost a 'love at first bite' - the per...
```

```
In [13]: final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

```
In [14]: #Before starting the next phase of preprocessing lets see the number of entries left
         print(final.shape)

         #How many positive and negative reviews are present in our dataset?
         final['Score'].value_counts()
```

```
(71551, 10)
```

```
Out[14]: 1    59256
         0    12295
         Name: Score, dtype: int64
```

4 [3] Preprocessing

4.1 [3.1]. Preprocessing Review Text

Now that we have finished deduplication our data requires some preprocessing before we go on further with analysis and making the prediction model.

Hence in the Preprocessing phase we do the following in the order below:-

1. Begin by removing the html tags
2. Remove any punctuations or limited set of special characters like , or . or # etc.
3. Check if the word is made up of english letters and is not alpha-numeric
4. Check to see if the length of the word is greater than 2 (as it was researched that there is no adjective in 2-letters)
5. Convert the word to lowercase
6. Remove Stopwords
7. Finally Snowball Stemming the word (it was obsereved to be better than Porter Stemming)

After which we collect the words used to describe positive and negative reviews

In [15]:
```
# printing some random reviews
sent_0 = final['Text'].values[0]
print(sent_0)
print("="*50)

sent_1000 = final['Text'].values[1000]
print(sent_1000)
print("="*50)

sent_1500 = final['Text'].values[1500]
print(sent_1500)
print("="*50)

sent_4900 = final['Text'].values[4900]
print(sent_4900)
print("="*50)
```

```
A charming, rhyming book that describes the circumstances under which you eat (or don't) chick
==================================================
I have one cup a day and it really decreases my night sweats. I'm in amazement at how much it h
==================================================
Strong without being bitter, this is my favorite tea by far.  I was pleasantly surprised recent
==================================================
The Kay's Naturals protein crispy Parmesan  pack of 12 protein chips tasted aweful. I would not
==================================================
```

In [16]:
```
# remove urls from text python: https://stackoverflow.com/a/40823105/4084039
sent_0 = re.sub(r"http\S+", "", sent_0)
sent_1000 = re.sub(r"http\S+", "", sent_1000)
sent_150 = re.sub(r"http\S+", "", sent_1500)
sent_4900 = re.sub(r"http\S+", "", sent_4900)

print(sent_0)
```

```
A charming, rhyming book that describes the circumstances under which you eat (or don't) chick
```

In [17]:
```
# https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-all
from bs4 import BeautifulSoup

soup = BeautifulSoup(sent_0, 'lxml')
text = soup.get_text()
print(text)
print("="*50)

soup = BeautifulSoup(sent_1000, 'lxml')
text = soup.get_text()
print(text)
print("="*50)

soup = BeautifulSoup(sent_1500, 'lxml')
text = soup.get_text()
print(text)
print("="*50)

soup = BeautifulSoup(sent_4900, 'lxml')
text = soup.get_text()
print(text)
```

```
A charming, rhyming book that describes the circumstances under which you eat (or don't) chick
==================================================
I have one cup a day and it really decreases my night sweats. I'm in amazement at how much it h
==================================================
Strong without being bitter, this is my favorite tea by far.  I was pleasantly surprised recen
==================================================
The Kay's Naturals protein crispy Parmesan  pack of 12 protein chips tasted aweful. I would not
```

In [18]:
```
# https://stackoverflow.com/a/47091490/4084039
import re

def decontracted(phrase):
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase
```

In [19]:
```
sent_1500 = decontracted(sent_1500)
print(sent_1500)
print("="*50)
```

```
Strong without being bitter, this is my favorite tea by far.  I was pleasantly surprised recen
==================================================
```

In [20]:
```
#remove words with numbers python: https://stackoverflow.com/a/18082370/4084039
sent_0 = re.sub("\S*\d\S*", "", sent_0).strip()
print(sent_0)
```

```
A charming, rhyming book that describes the circumstances under which you eat (or don't) chick
```

In [21]:
```
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent_1500 = re.sub('[^A-Za-z0-9]+', ' ', sent_1500)
print(sent_1500)
```

```
Strong without being bitter this is my favorite tea by far I was pleasantly surprised recently
```

In [22]:
```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
# <br /><br /> ==> after the above steps, we are getting "br br"
# we are including them into stop words list
# instead of <br /> if we have <br/> these tags would have revmoved in the 1st step

stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselve
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him'
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself',
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', '
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'a
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'throug
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', '
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'a
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'to
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", '
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mi
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't",
            'won', "won't", 'wouldn', "wouldn't"])
```

In [23]:
```
# Combining all the above stundents
from tqdm import tqdm
preprocessed_reviews = []
# tqdm is for printing the status bar
for sentance in tqdm(final['Text'].values):
```

```
        sentance = re.sub(r"http\S+", "", sentance)
        sentance = BeautifulSoup(sentance, 'lxml').get_text()
        sentance = decontracted(sentance)
        sentance = re.sub("\S*\d\S*", "", sentance).strip()
        sentance = re.sub('[^A-Za-z]+', ' ', sentance)
        # https://gist.github.com/sebleier/554280
        sentance = ' '.join(e.lower() for e in sentance.split() if e.lower() not in stopwo
        preprocessed_reviews.append(sentance.strip())
```

```
100%|| 71551/71551 [00:27<00:00, 2592.95it/s]
```

```
In [24]: preprocessed_reviews[1500]
```

```
Out[24]: 'strong without bitter favorite tea far pleasantly surprised recently showed productio
```

[3.2] Preprocessing Review Summary

```
In [25]: ## Similartly you can do preprocessing for review summary also.
```

5 [4] Featurization

5.1 [4.1] BAG OF WORDS

```
In [28]: #BoW
         #min_df=5 ?.
         count_vect = CountVectorizer() #in scikit-learn
         count_vect.fit(preprocessed_reviews)
         print("some feature names ", count_vect.get_feature_names()[:10])
         print('='*50)

         final_counts = count_vect.transform(preprocessed_reviews)
         print("the type of count vectorizer ",type(final_counts))
         print("the shape of out text BOW vectorizer ",final_counts.get_shape())
         print("the number of unique words ", final_counts.get_shape()[1])
```

```
some feature names  ['aa', 'aaa', 'aaaa', 'aaaaa', 'aaaaaaah', 'aaaaaahhhhh', 'aaaaallll', 'aaa
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (71551, 50522)
the number of unique words  50522
```

5.2 [4.2] Bi-Grams and n-Grams.

```
In [29]: #bi-gram, tri-gram and n-gram

         #removing stop words like "not" should be avoided before building n-grams
         # count_vect = CountVectorizer(ngram_range=(1,2))
```

```
# please do read the CountVectorizer documentation http://scikit-learn.org/stable/mod

# you can choose these numebrs min_df=10, max_features=5000, of your choice
count_vect = CountVectorizer(ngram_range=(1,2), min_df=10, max_features=5000)
final_bigram_counts = count_vect.fit_transform(preprocessed_reviews)
print("the type of count vectorizer ",type(final_bigram_counts))
print("the shape of out text BOW vectorizer ",final_bigram_counts.get_shape())
print("the number of unique words including both unigrams and bigrams ", final_bigram_
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (71551, 5000)
the number of unique words including both unigrams and bigrams  5000
```

5.3 [4.3] TF-IDF

In [30]:
```
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
tf_idf_vect.fit(preprocessed_reviews)
print("some sample features(unique words in the corpus)",tf_idf_vect.get_feature_names
print('='*50)

final_tf_idf = tf_idf_vect.transform(preprocessed_reviews)
print("the type of count vectorizer ",type(final_tf_idf))
print("the shape of out text TFIDF vectorizer ",final_tf_idf.get_shape())
print("the number of unique words including both unigrams and bigrams ", final_tf_idf
```

```
some sample features(unique words in the corpus) ['aa', 'aafco', 'aback', 'abandoned', 'abc',
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (71551, 41692)
the number of unique words including both unigrams and bigrams  41692
```

5.4 [4.4] Word2Vec

In [0]:
```
# Train your own Word2Vec model using your own text corpus
i=0
list_of_sentance=[]
for sentance in preprocessed_reviews:
    list_of_sentance.append(sentance.split())
```

In [0]:
```
# Using Google News Word2Vectors

# in this project we are using a pretrained model by google
# its 3.3G file, once you load this into your memory
# it occupies ~9Gb, so please do this step only if you have >12G of ram
# we will provide a pickle file wich contains a dict ,
# and it contains all our courpus words as keys and  model[word] as values
# To use this code-snippet, download "GoogleNews-vectors-negative300.bin"
```

```
# from https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit
# it's 1.9GB in size.


# http://kavita-ganesan.com/gensim-word2vec-tutorial-starter-code/#.W17SRFAzZPY
# you can comment this whole cell
# or change these varible according to your need

is_your_ram_gt_16g=False
want_to_use_google_w2v = False
want_to_train_w2v = True

if want_to_train_w2v:
    # min_count = 5 considers only words that occured atleast 5 times
    w2v_model=Word2Vec(list_of_sentance,min_count=5,size=50, workers=4)
    print(w2v_model.wv.most_similar('great'))
    print('='*50)
    print(w2v_model.wv.most_similar('worst'))

elif want_to_use_google_w2v and is_your_ram_gt_16g:
    if os.path.isfile('GoogleNews-vectors-negative300.bin'):
        w2v_model=KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300.bin
        print(w2v_model.wv.most_similar('great'))
        print(w2v_model.wv.most_similar('worst'))
    else:
        print("you don't have gogole's word2vec file, keep want_to_train_w2v = True, to
```

```
[('snack', 0.9951335191726685), ('calorie', 0.9946465492248535), ('wonderful', 0.99460321664810
==================================================
[('varieties', 0.9994194507598877), ('become', 0.9992934465408325), ('popcorn', 0.999275088310
```

In [0]:
```
w2v_words = list(w2v_model.wv.vocab)
print("number of words that occured minimum 5 times ",len(w2v_words))
print("sample words ", w2v_words[0:50])
```

```
number of words that occured minimum 5 times  3817
sample words  ['product', 'available', 'course', 'total', 'pretty', 'stinky', 'right', 'nearby
```

5.5 [4.4.1] Converting text into vectors using Avg W2V, TFIDF-W2V

[4.4.1.1] Avg W2v

In [0]:
```
# average Word2Vec
# compute average word2vec for each review.
sent_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(list_of_sentance): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need to
```

```
            cnt_words =0; # num of words with a valid vector in the sentence/review
            for word in sent: # for each word in a review/sentence
                if word in w2v_words:
                    vec = w2v_model.wv[word]
                    sent_vec += vec
                    cnt_words += 1
            if cnt_words != 0:
                sent_vec /= cnt_words
            sent_vectors.append(sent_vec)
        print(len(sent_vectors))
        print(len(sent_vectors[0]))
```

```
100%|| 4986/4986 [00:03<00:00, 1330.47it/s]


4986
50
```

[4.4.1.2] TFIDF weighted W2v

```
In [0]: # S = ["abc def pqr", "def def def abc", "pqr pqr def"]
        model = TfidfVectorizer()
        tf_idf_matrix = model.fit_transform(preprocessed_reviews)
        # we are converting a dictionary with word as a key, and the idf as a value
        dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

```
In [0]: # TF-IDF weighted Word2Vec
        tfidf_feat = model.get_feature_names() # tfidf words/col-names
        # final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfidf

        tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this li
        row=0;
        for sent in tqdm(list_of_sentance): # for each review/sentence
            sent_vec = np.zeros(50) # as word vectors are of zero length
            weight_sum =0; # num of words with a valid vector in the sentence/review
            for word in sent: # for each word in a review/sentence
                if word in w2v_words and word in tfidf_feat:
                    vec = w2v_model.wv[word]
        #             tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
                    # to reduce the computation we are
                    # dictionary[word] = idf value of word in whole courpus
                    # sent.count(word) = tf valeus of word in this review
                    tf_idf = dictionary[word]*(sent.count(word)/len(sent))
                    sent_vec += (vec * tf_idf)
                    weight_sum += tf_idf
            if weight_sum != 0:
                sent_vec /= weight_sum
```

```
            tfidf_sent_vectors.append(sent_vec)
            row += 1
```

```
100%|| 4986/4986 [00:20<00:00, 245.63it/s]
```

6 [5] Assignment 4: Apply Naive Bayes

```
<li><strong>Apply Multinomial NaiveBayes on these feature sets</strong>
    <ul>
        <li><font color='red'>SET 1:</font>Review text, preprocessed one converted into vectors
        <li><font color='red'>SET 2:</font>Review text, preprocessed one converted into vectors
    </ul>
</li>
<br>
<li><strong>The hyper paramter tuning(find best Alpha)</strong>
    <ul>
<li>Find the best hyper parameter which will give the maximum <a href='https://www.appliedaico
<li>Consider a wide range of alpha values for hyperparameter tuning, start as low as 0.00001</l
<li>Find the best hyper paramter using k-fold cross validation or simple cross validation data
<li>Use gridsearch cv or randomsearch cv or you can also write your own for loops to do this ta
    </ul>
</li>
<br>
<li><strong>Feature importance</strong>
    <ul>
<li>Find the top 10 features of positive class and top 10 features of negative class for both
    </ul>
</li>
<br>
<li><strong>Feature engineering</strong>
    <ul>
<li>To increase the performance of your model, you can also experiment with with feature engine
        <ul>
        <li>Taking length of reviews as another feature.</li>
        <li>Considering some features from review summary as well.</li>
    </ul>
    </ul>
</li>
<br>
<li><strong>Representation of results</strong>
    <ul>
<li>You need to plot the performance of model both on train data and cross validation data for
<img src='train_cv_auc.JPG' width=300px></li>
<li>Once after you found the best hyper parameter, you need to train your model with it, and fi
<img src='train_test_auc.JPG' width=300px></li>
<li>Along with plotting ROC curve, you need to print the <a href='https://www.appliedaicourse.c
```

```
<img src='confusion_matrix.png' width=300px></li>
    </ul>
</li>
<br>
<li><strong>Conclusion</strong>
    <ul>
<li>You need to summarize the results at the end of the notebook, summarize it in the table fo
    <img src='summary.JPG' width=400px>
</li>
    </ul>
```

Note: Data Leakage

1. There will be an issue of data-leakage if you vectorize the entire data and then split it into train/cv/test.
2. To avoid the issue of data-leakag, make sure to split your data first and then vectorize it.
3. While vectorizing your data, apply the method fit_transform() on you train data, and apply the method transform() on cv/test data.
4. For more details please go through this link.

7 Applying Multinomial Naive Bayes

In [26]:
```
Y = final['Score']
X = preprocessed_reviews
#make sure we have correct X and Y
print(Y.shape)
print(len(X))

# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_tes
from sklearn.model_selection import train_test_split
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sk
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import confusion_matrix
from tqdm import tqdm

from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
import math

import seaborn as sns
import matplotlib.pyplot as plt


# Train on oldest data (eg. Now - 90 days), CV on somewhat recent  data (eg. Now - 30
# doing a time series split:  swapped the test and train as our data is in DESCENDING
# and we want X_test to have the most recent data.
X_test, X_train, y_test, y_train = train_test_split(X, Y, test_size=0.77, shuffle=Fal
```

```
# time series splitting
# not splitting further into CV and Train because we plan to use GridSearch with 3 fo
# we pas in entire X_train to GridSearch.\
#X_cv, X_train, y_cv, y_train = train_test_split(X_train, y_train, test_size=0.77, sh
# do random between train and CV
#X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33)

print('X_train size=' , len(X_train))
print('X_test size=', len(X_test))
print('y_train class counts')
print(y_train.value_counts())
print('y_test class counts')
print(y_test.value_counts())

#2. Apply Multinomial Naive Bayes on two feature sets, one with BOW and other with TF
#3. Consider a minimum of 100k points for this assignment as the model runs very quic
#4. Use AUC as a metric for hyperparameter tuning. And take the range of alpha values
#5. Find the top 10 features of positive class and top 10 features of negative class
#note: please do submit both .pdf and .ipynb versions of your notebook
#6. If you want to further increase the performance of the model, you can experiment
```

```
(71551,)
71551
X_train size= 55095
X_test size= 16456
y_train class counts
1    45830
0     9265
Name: Score, dtype: int64
y_test class counts
1    13426
0     3030
Name: Score, dtype: int64
```

In [27]:
```
def predictAndPlot(X_train, y_train, X_test, y_test, clf):
    train_fpr, train_tpr, thresholds = roc_curve(y_train, clf.predict_proba(X_train)[
    test_fpr, test_tpr, thresholds = roc_curve(y_test, clf.predict_proba(X_test)[:,1]
    plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr))
    plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
    plt.legend()
    plt.xlabel("Alpha: hyperparameter")
    plt.ylabel("AUC")
    plt.title("ERROR PLOTS")
    plt.show()
    print("="*100)
```

```
    cmTrain = confusion_matrix(y_train, clf.predict(X_train))
    #print("Test confusion matrix")
    cmTest= confusion_matrix(y_test, clf.predict(X_test))



    plt.figure(figsize=(10,5))
    trainx= plt.subplot(1, 3, 1)
    sns.heatmap(cmTrain, annot=True, ax = trainx, cmap='Blues', fmt='g'); #annot=True
    # labels, title and ticks
    trainx.set_xlabel('Actual');trainx.set_ylabel('Predicted');
    trainx.set_title('Train Confusion Matrix');
    trainx.xaxis.set_ticklabels(['negative', 'positive']); trainx.yaxis.set_ticklabels

    testx= plt.subplot(1, 3, 3)
    sns.heatmap(cmTest, annot=True, ax = testx, cmap='Blues', fmt='g'); #annot=True t
    # labels, title and ticks
    testx.set_xlabel('Actual');testx.set_ylabel('Predicted');
    testx.set_title('Test Confusion Matrix');
    testx.xaxis.set_ticklabels(['negative', 'positive']); testx.yaxis.set_ticklabels(

def Sort_TupleList(tupleList):

    # getting length of list of tuples
    lst = len(tupleList)
    for i in range(0, lst):
        for j in range(0, lst-i-1):
            if (tupleList[j][1] < tupleList[j + 1][1]):
                temp = tupleList[j]
                tupleList[j]= tupleList[j + 1]
                tupleList[j + 1]= temp
    return tupleList
```

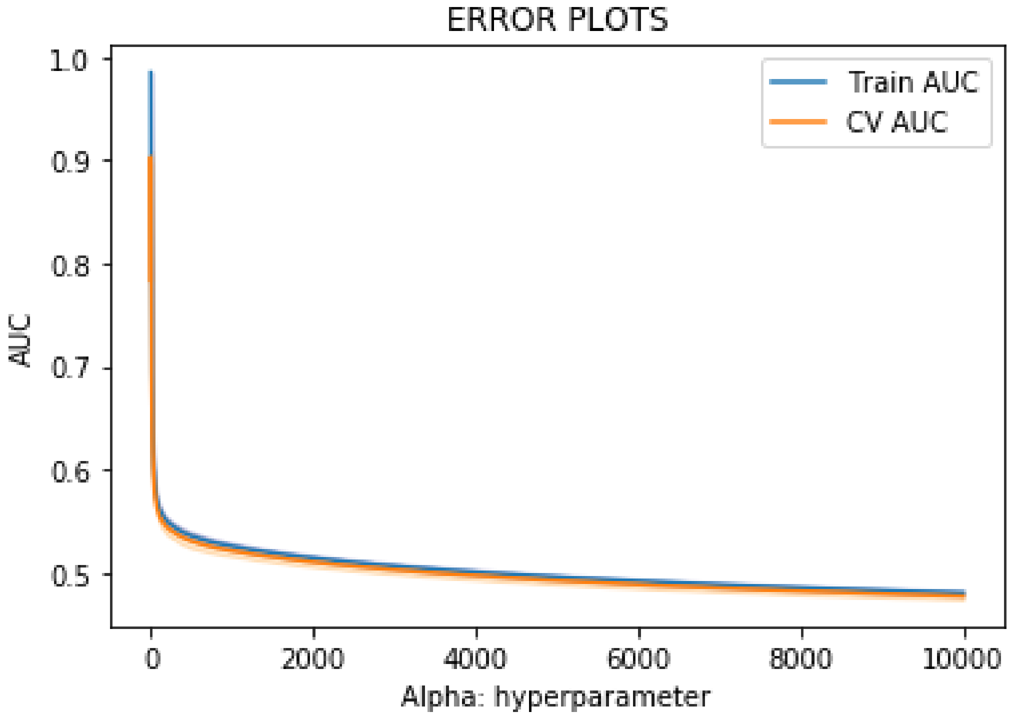
7.1 [5.1] Applying Naive Bayes on BOW, SET 1

In [28]:
```
# Please write all the code with proper documentation
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearc
from sklearn.model_selection import GridSearchCV
import numpy


# missed providing min_df
vectorizer = CountVectorizer()

# While vectorizing your data, apply the method fit_transform() on you train data,
```

```
    # and apply the method transform() on cv/test data.
    # THE VOCABULARY SHOULD BUILT ONLY WITH THE WORDS OF TRAIN DATA
    vectorizer.fit(X_train)

    # we use the fitted CountVectorizer to convert the text to vector
    X_train_bow = vectorizer.transform(X_train)
    X_test_bow = vectorizer.transform(X_test)

    print("After vectorizations")
    print(X_train_bow.shape, y_train.shape)
    print(X_test_bow.shape, y_test.shape)
    print(type(X_train_bow))

    nb = MultinomialNB()
    a = [0.00001, 0.00005, 0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.2, 0.3, 0.4, 
    b = [x for x in range(1, 10000, 1)]
    Z = a + b
    parameters = {'alpha':Z}
    clf = GridSearchCV(nb, parameters, cv=3, scoring='roc_auc')
    clf.fit(X_train_bow, y_train)

    train_auc= clf.cv_results_['mean_train_score']
    train_auc_std= clf.cv_results_['std_train_score']
    cv_auc = clf.cv_results_['mean_test_score']
    cv_auc_std= clf.cv_results_['std_test_score']
```

```
After vectorizations
(55095, 43816) (55095,)
(16456, 43816) (16456,)
<class 'scipy.sparse.csr.csr_matrix'>
```

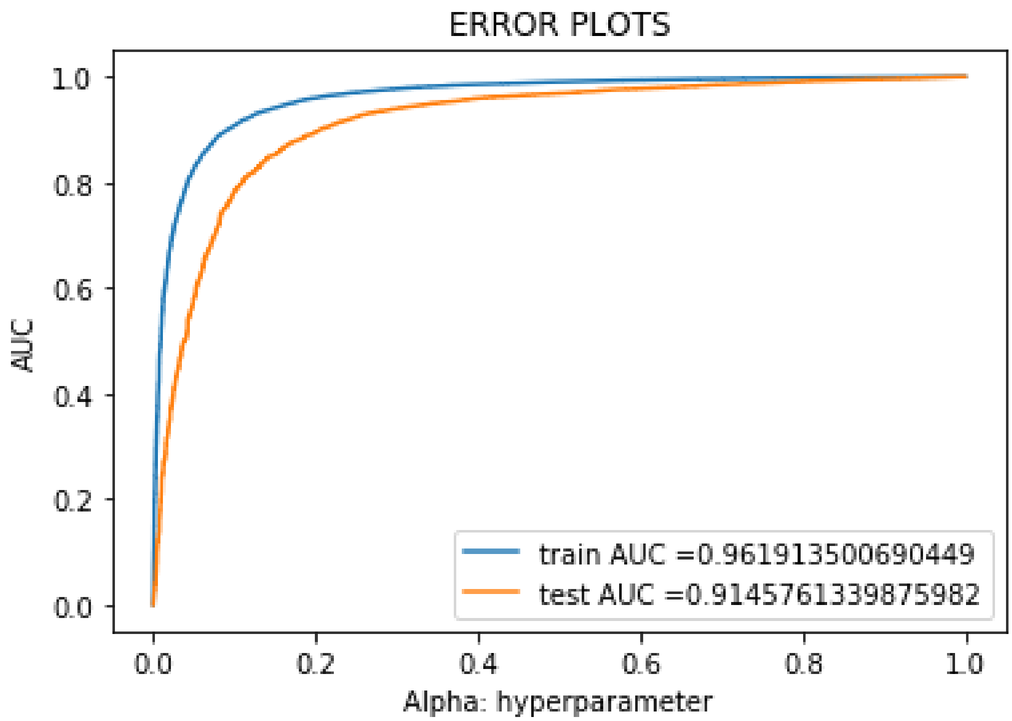
In [29]:
```
plt.plot(Z, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(Z,train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.

plt.plot(Z, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(Z,cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='dar
plt.legend()
plt.xlabel("Alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



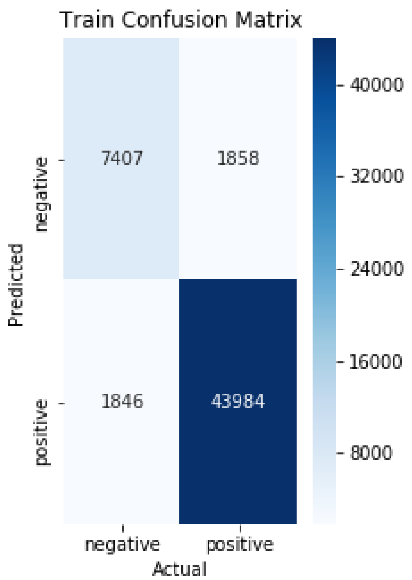

```
In [30]: from sklearn.metrics import confusion_matrix
         y_pred = clf.predict(X_test_bow)
         print('Confusion Matrix : \n' + str(confusion_matrix(y_test,y_pred)))
```

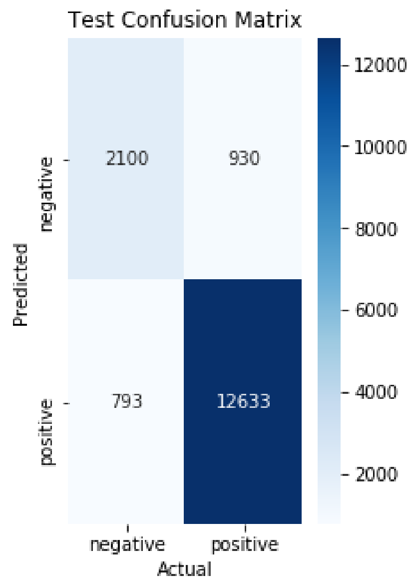
```
Confusion Matrix :
[[ 2100   930]
 [  793 12633]]
```

```
In [31]: #The .best_estimator_ attribute of the Wrapped GridSearchCV classifier is an instance
         #which has the 'best' combination of given parameters from the param_grid.
         #So directly using the GridSerachCV classfier here as It just uses the same best esti
         predictAndPlot(X_train_bow, y_train, X_test_bow, y_test, clf)
```


ERROR PLOTS
AUC
Alpha: hyperparameter
train AUC =0.961913500690449
test AUC =0.9145761339875982

==


Train Confusion Matrix
Predicted
negative
positive
Actual
7407
1858
1846
43984


Test Confusion Matrix
Predicted
negative
positive
Actual
2100
930
793
12633

8 Feature Vs Likelihood Computations : P(x_i|y=class)

In [32]:
```
bestAlpha = clf.best_params_
print(bestAlpha)
#Find the top 10 features of positive class and top 10 features of negative class
#for both feature sets Set 1 and Set 2 using values of feature_log_prob_ parameter
#of MultinomialNB and print their corresponding feature names
feature_probs = clf.best_estimator_.feature_log_prob_
print(type(feature_probs))
print(feature_probs.shape)
positive = feature_probs[1]
negative = feature_probs[0]
feature_names = vectorizer.get_feature_names()
print('feature_names count=', len(feature_names))
print('positive class feature count=', len(positive))
print('negative class feature count=', len(negative))

positive_tuples = list(zip(feature_names, positive))
negative_tuples = list(zip(feature_names, negative))

#print(positive_tuples)
#print(negative_tuples)
#positiveDescending = np.sort(positive)[::-1]
#print(positiveDescending)
#negativeDescending = np.sort(negative)[::-1]
#print(negativeDescending)
```

```
{'alpha': 0.4}
<class 'numpy.ndarray'>
(2, 43816)
feature_names count= 43816
positive class feature count= 43816
negative class feature count= 43816
```

8.0.1 [5.1.1] Top 10 important features of positive class from SET 1

In [33]:
```
sorted_positive_tuples = Sort_TupleList(positive_tuples)
print(sorted_positive_tuples[:10])
```

```
[('not', -3.713009991850404), ('like', -4.517782777010522), ('good', -4.661055696144963), ('gr
```

8.0.2 [5.1.2] Top 10 important features of negative class from SET 1

In [34]:
```
sorted_negative_tuples = Sort_TupleList(negative_tuples)
print(sorted_negative_tuples[:10])
```

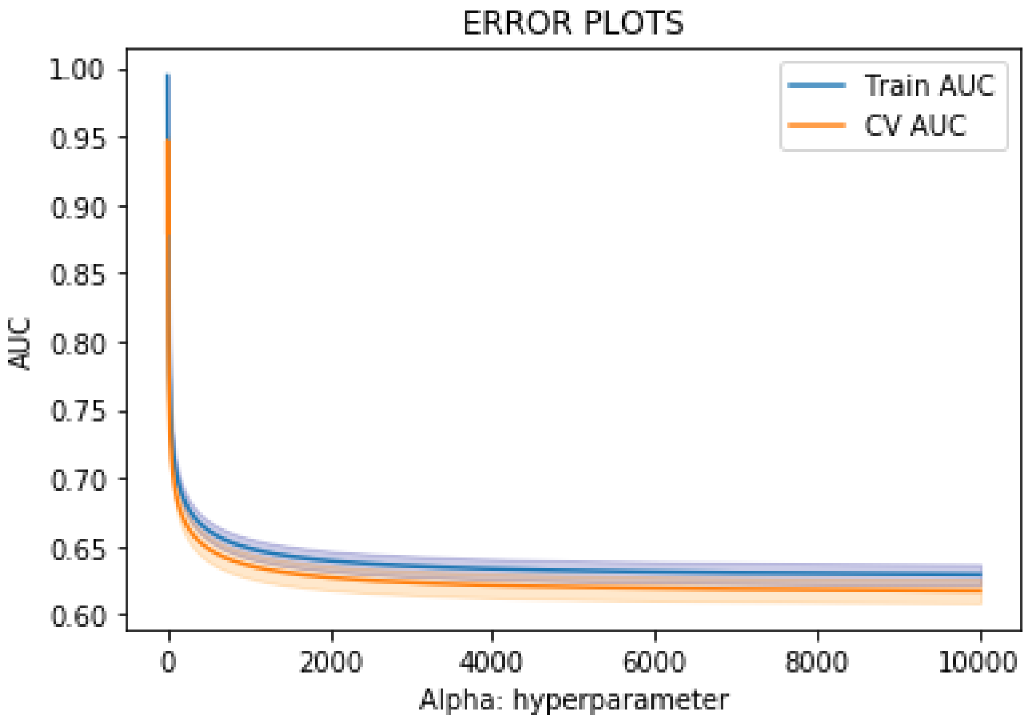
```
[('not', -3.2947422663612844), ('like', -4.452917738633204), ('product', -4.561982482156642),
```

8.1 [5.2] Applying Naive Bayes on TFIDF, SET 2

In [39]:
```
# Please write all the code with proper documentation
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer
from tqdm import tqdm
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt


vectorizer = TfidfVectorizer(ngram_range=(1,2), min_df=10)
vectorizer.fit(X_train)

# we use the fitted vectorizer to convert the text to vector
X_train_tfidf = vectorizer.transform(X_train)
X_test_tfidf = vectorizer.transform(X_test)

print("After vectorizations")
print(X_train_tfidf.shape, y_train.shape)
print(X_test_tfidf.shape, y_test.shape)
print(type(X_train_tfidf))

nb = MultinomialNB()
a = [0.00001, 0.00005, 0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.2, 0.3, 0.4, 0
b = [x for x in range(1, 10000, 1)]
Z = a + b

parameters = {'alpha':Z}
clf = GridSearchCV(nb, parameters, cv=3, scoring='roc_auc')
clf.fit(X_train_tfidf, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

plt.plot(Z, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(Z,train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.

plt.plot(Z, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(Z,cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='dar
plt.legend()
plt.xlabel("Alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
```

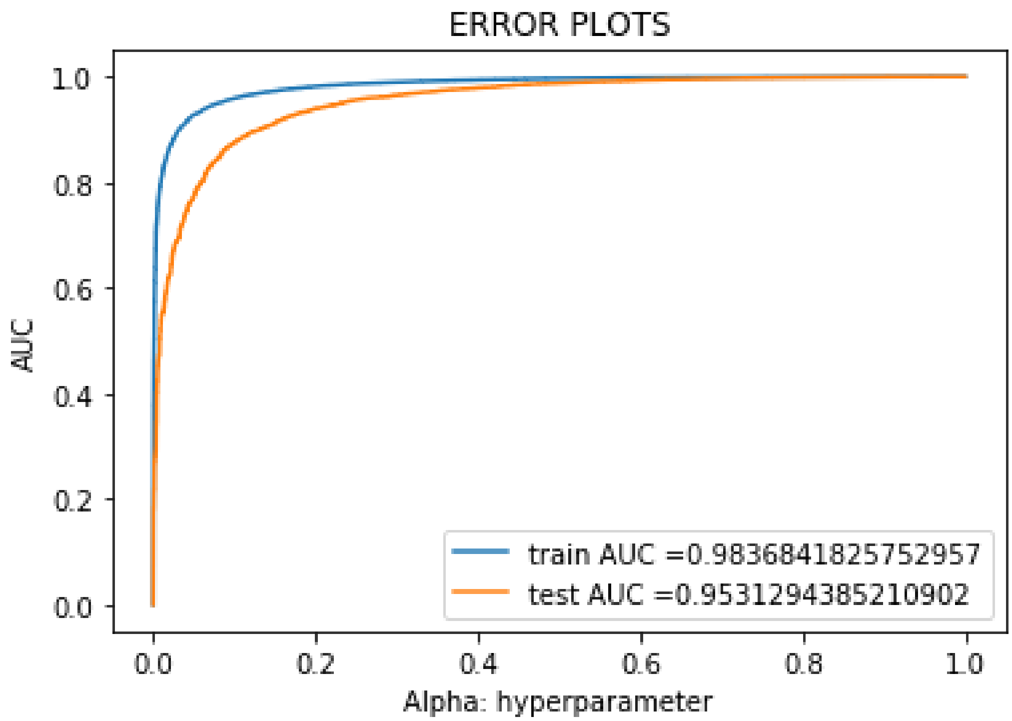
```
        plt.show()
```

```
After vectorizations
(55095, 32378) (55095,)
(16456, 32378) (16456,)
<class 'scipy.sparse.csr.csr_matrix'>
```

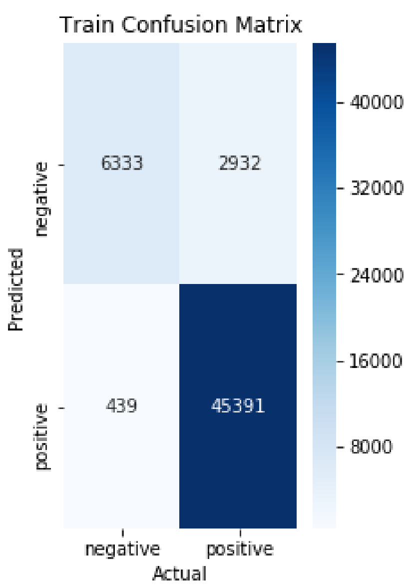


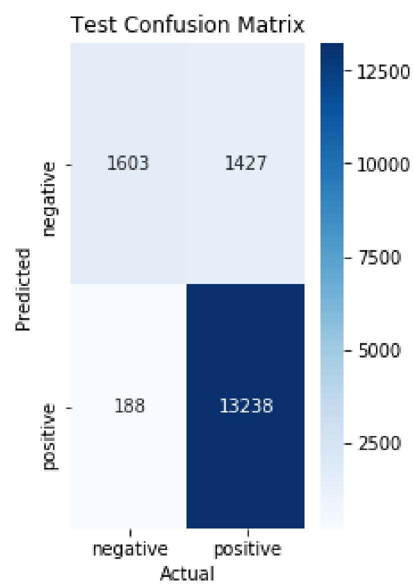

```
In [40]: from sklearn.metrics import confusion_matrix
         y_pred = clf.predict(X_test_tfidf)
         print('Confusion Matrix : \n' + str(confusion_matrix(y_test,y_pred)))

         predictAndPlot(X_train_tfidf, y_train, X_test_tfidf, y_test, clf)
```

```
Confusion Matrix :
[[ 1603  1427]
 [  188 13238]]
```


ERROR PLOTS
AUC
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
Alpha: hyperparameter
train AUC =0.9836841825752957
test AUC =0.9531294385210902

==


Train Confusion Matrix
Predicted
negative
positive
6333
2932
439
45391
40000
32000
24000
16000
8000
negative
positive
Actual


Test Confusion Matrix
Predicted
negative
positive
1603
1427
188
13238
12500
10000
7500
5000
2500
negative
positive
Actual

9 Feature Vs Likelihood Computations : P(x_i | y=class)

```
In [41]: bestAlpha = clf.best_params_
         print(bestAlpha)
         #Find the top 10 features of positive class and top 10 features of negative class
         #for both feature sets Set 1 and Set 2 using values of feature_log_prob_ parameter
         #of MultinomialNB and print their corresponding feature names
         feature_probs = clf.best_estimator_.feature_log_prob_
         print(type(feature_probs))
         print(feature_probs.shape)
         positive = feature_probs[1]
         negative = feature_probs[0]
         feature_names = vectorizer.get_feature_names()
         print('feature_names count=', len(feature_names))
         print('positive class feature count=', len(positive))
         print('negative class feature count=', len(negative))

         positive_tuples = list(zip(feature_names, positive))
         negative_tuples = list(zip(feature_names, negative))
```

```
{'alpha': 0.1}
<class 'numpy.ndarray'>
(2, 32378)
feature_names count= 32378
positive class feature count= 32378
negative class feature count= 32378
```

9.0.1 [5.2.1] Top 10 important features of positive class from SET 2

```
In [42]: sorted_positive_tuples = Sort_TupleList(positive_tuples)
         print(sorted_positive_tuples[:10])
```

```
[('not', -5.281133289848212), ('great', -5.615339333251361), ('good', -5.68539163649578), ('co
```

9.0.2 [5.2.2] Top 10 important features of negative class from SET 2

```
In [43]: sorted_negative_tuples = Sort_TupleList(negative_tuples)
         print(sorted_negative_tuples[:10])
```

```
[('not', -4.728564349862339), ('product', -5.553935249733819), ('like', -5.586624940176788), (
```

10 [6] Conclusions

```
In [44]: from prettytable import PrettyTable
```

```
x = PrettyTable()
x.field_names = ["Vectorizer", "Algorithm", "HyperParameter", "AUC", "DataSize","min_c
x.add_row(["BOW", "NB", 1.00001, 0.91, "100k", 1])
x.add_row(["TFIDF", "NB", 1.00001, 0.95, "100k", 10])

print("Tabular Results:")
print()
print()
print(x)
```

```
Tabular Results:


+------------+-----------+----------------+------+----------+------------------+
| Vectorizer | Algorithm | HyperParameter | AUC  | DataSize | min_count/min_df |
+------------+-----------+----------------+------+----------+------------------+
|    BOW     |     NB    |    1.00001     | 0.91 |   100k   |        1         |
|   TFIDF    |     NB    |    1.00001     | 0.95 |   100k   |        10        |
+------------+-----------+----------------+------+----------+------------------+
```

Observations: 1. Naive Bayes is able to handle large number of features for Text Classification and the results are better than KNN. It also consumes less time than KNN thereby permitting one to apply GridSearch for hyperparameter tuning. 2. TFIDF with 1-gram and 2-gram and a min-df of 10 performs better than Bag-Of-Words.